



Entire Screen Builder

| Version 5.2.1 | Script Files



This document applies to Entire Screen Builder Version 5.2.1 and to all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

© Copyright Software AG 1999-2003
All rights reserved.

The name Software AG and/or all Software AG product names are either trademarks or registered trademarks of Software AG. Other company and product names mentioned herein may be trademarks of their respective owners.

Table of Contents

Script Files	1
Script Files	1
General Information on Script Files	2
General Information on Script Files	2
What is a Script File?	2
Script File Folders	2
Startup Scripts	3
Using Backslashes	3
Example Script File	4
Running Script Files in the Development Environment	4
The NSW Object	5
The NSW Object	5
Syntax Conventions	5
NSW Object Properties	6
NSW Object Methods	6
User Variables	7
Error Codes	7
Display Status	9
Windows Message Box Flags	9
Windows Button Identifiers	9
Host Communication Methods	10
Host Communication Methods	10
CloseSession	11
Syntax	11
Parameters	11
Description	11
Example	11
Return Values	11
GetDefaultSession	12
Syntax	12
Parameters	12
Description	12
Example	12
Return Values	12
GetDisplay	13
Syntax	13
Parameters	13
Description	13
Example	13
Return Values	13
GetScreenText	14
Syntax	14
Parameters	14
Description	14
Example	14
Return Values	14
OpenSession	15
Syntax	15

Parameters	15
Description	15
Examples	15
Return Values	15
OpenUpdatedSession	16
Syntax	16
Parameters	16
Description	16
Example	16
Return Values	16
SendKey	17
Syntax	17
Parameters	17
Description	17
Examples	17
Return Values	17
SetDataTransfer	18
Syntax	18
Parameters	18
Description	18
Example	18
Return Values	18
SetDisplay	19
Syntax	19
Parameters	19
Description	19
Example	19
Return Values	19
SetSessionParameter	20
Syntax	20
Parameters	20
Description	20
Examples	22
Return Values	22
SetUnixLogonCredentials	23
Syntax	23
Parameters	23
Description	23
Example	23
Return Values	23
SetXMLEncoding	24
Syntax	24
Parameters	24
Description	24
Examples	24
Return Values	24
SetXMLStyleSheet	25
Syntax	25
Parameters	25
Description	25
Examples	25

Return Values	25
TypeString	26
Syntax	26
Parameters	26
Description	26
Examples	26
Return Values	26
UseDefinedSession	27
Syntax	27
Parameters	27
Description	27
Example	27
Return Values	27
User Interface Methods	28
User Interface Methods	28
Cancel	29
Syntax	29
Parameters	29
Description	29
Example	29
Return Values	29
DeleteUserVariable	30
Syntax	30
Parameters	30
Description	30
Example	30
Return Values	30
GetClientArgument	31
Syntax	31
Parameters	31
Description	31
Example	31
Return Values	31
GetUserConnectionValue	32
Syntax	32
Parameters	32
Description	32
Example	32
Return Values	32
GetUserInput	33
Syntax	33
Parameters	33
Description	33
Example	33
Return Values	33
GetUserVariableValue	34
Syntax	34
Parameters	34
Description	34
Example	34
Return Values	34

MessageBox	35
Syntax	35
Parameters	35
Description	35
Example	35
Return Values	35
OutputMsg	36
Syntax	36
Parameters	36
Description	36
Example	36
Return Values	36
SetClientArgument	37
Syntax	37
Parameters	37
Description	37
Example	37
Return Values	37
SetUserConnectionValue	38
Syntax	38
Parameters	38
Description	38
Example	38
Return Values	38
SetUserVariableValue	39
Syntax	39
Parameters	39
Description	39
Example	39
Return Values	39
Processing Methods	40
Processing Methods	40
CallExternalFunction	41
Syntax	41
Parameters	41
Description	41
Example	42
Return Values	42
Pause	43
Syntax	43
Parameters	43
Description	43
Example	43
Return Values	43
Client-Side Access Methods	44
Client-Side Access Methods	44
BrowseClientFile	45
Syntax	45
Parameters	45
Description	45
Examples	45

Return Values	45
CloseClientFile	46
Syntax	46
Parameters	46
Description	46
Example	46
Return Values	46
CreateClientFolder	47
Syntax	47
Parameters	47
Description	47
Example	47
Return Values	47
DeleteClientFile	48
Syntax	48
Parameters	48
Description	48
Example	48
Return Values	48
DeleteClientFolder	49
Syntax	49
Parameters	49
Description	49
Example	49
Return Values	49
ExecuteClientApplication	50
Syntax	50
Parameters	50
Description	50
Example	50
Return Values	50
ListClientDirectory	51
Syntax	51
Parameters	51
Description	51
Example	51
Return Values	51
OpenClientFile	52
Syntax	52
Parameters	52
Description	52
Example	52
Return Values	52
ReadClientFile	53
Syntax	53
Parameters	53
Description	53
Example	53
Return Values	53
SetClientFilePtr	54
Syntax	54

Parameters	54
Description	54
Example	54
Return Values	54
SetCurrentClientFolder	55
Syntax	55
Parameters	55
Description	55
Example	55
Return Values	55
WriteClientFile	56
Syntax	56
Parameters	56
Description	56
Example	56
Return Values	56

Script Files

Script files for Entire Screen Builder are created using the JavaScript language. Entire Screen Builder supports the core JavaScript version 1.2 features and adds an Entire Screen Builder specific object to the language.

This documentation only explains the methods specific to Entire Screen Builder. For detailed information on JavaScript, see your JavaScript documentation. You can find such documentation, for example, on <http://developer.netscape.com/docs/manuals/communicator/jsref/contents.htm>.

Note that the client-side and server-side JavaScript objects are not part of the core JavaScript.

See also: *Scripting, User Exits and APIs* in *Introducing Entire Screen Builder*.

This documentation is organized under the following headings:

- General Information on Script Files How to create a script file. Information on required script file folders, startup scripts, usage of backslashes and an example script file.
- The NSW Object Syntax conventions for the object named NSW which is added to the core JavaScript language, NSW object properties, general information on the NSW object methods and user variables, lists of all defined error codes, display status, Windows message box flags and Windows button identifiers.
- Host Communication Methods Detailed descriptions of all NSW object methods for managing the communication with the host (for example, open a session or send a terminal emulation key to the session).
- User Interface Methods Detailed descriptions of all NSW object methods for managing the user interface (for example, send a message to the viewer or set the value of a user-defined variable).
- Processing Methods Detailed descriptions of all NSW object methods for processing script files (for example, call a DLL function or pause processing of the script file).
- Client-Side Access Methods Detailed descriptions of all NSW object methods for accessing resources on a client workstation (for example, open a file on a client or execute an application on a client).

See also: *Working with Script Files* and *Unattended Workstation* in the *Utilities* documentation.

General Information on Script Files

This chapter covers the following topics:

- What is a Script File?
 - Script File Folders
 - Startup Scripts
 - Using Backslashes
 - Example Script File
 - Running Script Files in the Development Environment
-

What is a Script File?

Script files (extension *js*) initiate operations which are executed on the Entire Screen Builder Server. You can create script files with an ASCII editor on your PC. For example, you can create a script file which automates the process of logging on to a host system. All required logon methods are contained in the script file and are executed when the script file is invoked.

Important:

The extension *js* must be in lower-case when using the Entire Screen Builder Server on UNIX. We also recommend that you use lower-case for the script file name.

Script files can be used by all types of viewers (Windows Viewer, Web Viewer and Terminal Viewer).

Script File Folders

The script files are stored in a script files folder on the same machine on which the Entire Screen Builder Server is located. For a default installation, the script files folder is *\Entire Screen Builder 5\scripts*.

You can change the script files folder using the System Management Hub. See *Server Settings* in Entire Screen Builder's *System Management Hub* documentation.

The script files folder contains two subfolders: *production* and *test*. For normal operation, the script files must be stored in the *production* subfolder. If you want to test and debug script files, these files must be stored in the *test* subfolder. Note that debugging is only possible with the Terminal Viewer.

Sample script files are copied to your hard disk during installation. They can be found in the folder *\Entire Screen Builder 5\samples\samplescript*.

Startup Scripts

A startup script is executed after a user has logged on to the Entire Screen Builder Server. It is defined in the user profile using the System Management Hub. See *Users* in Entire Screen Builder's *System Management Hub* documentation.

- **GUI Viewers**

A startup script for a user can only be invoked when anonymous logon has been disabled and when the connection number 0 has been specified. A detailed description for these attributes can be found in the *Overview of Client Control Properties* which is part of the *User Exits* documentation.

- **Terminal Viewer**

The startup script is executed immediately after the user has logged on to the Entire Screen Builder Server.

Calling the startup script does not establish a connection between the viewer and the Entire Screen Builder Server. The connection to the server is only established after a successful connection to a host session using the communication methods `OpenSession` or `OpenUpdatedSession`. Due to this restriction, several script methods (which are partially executed in the viewer) can only be executed after a connection to a session has been established. Otherwise, `NSW(errno)` is set to `NSW_NO_OPEN_SESSION`. This applies to the user interface methods `MessageBox`, `OutputMsg`, `GetUserInput` and all client-side access methods.

Using Backslashes

As with most programming languages, the backslash has the special meaning of an escape character. Together with the character by which it is followed, the backslash defines a special character constant (for example, "`\n`" means "new line").

If the backslash is part of a string, you have to double it. Example:

```
Result = NSW.SetDataTransfer("SET PCFILE 7 DOWN DATA C:\\\\TEMP\\\\TEST.NCD");
```

You can also use one forward slash instead of two backslashes. Thus, the following statements are both valid:

```
dwErrorReturn = NSW.CreateClientFolder ("C:\\Myfolder")
dwErrorReturn = NSW.CreateClientFolder ("C:/Myfolder")
```

Example Script File

The following example shows how to logon to a VM mainframe system:

```
// logon.js: sample script file to logon to a mainframe system

if (NSW.GetScreenText(18,8,7) == "NETwork")
{
    NSW.TypeString(-1, -1, "demouser", TAB);
    Passwd = NSW.GetUserInput("Password-Request", "Please enter your password", PASSWORD_CHAR);
    NSW.TypeString(-1, -1, Passwd, CR);
}

bWait = true;

while (bWait == true)
{
    if (NSW.GetScreenText(5,3,5) != "Level")
        NSW.Pause(1000);
    else
        bWait = false;
}

NSW.SendKey(-1, -1, PF1);
NSW.Pause(2000);
if (NSW.GetScreenText(1,30,17) != "C O N - N E C T 3")
    NSW.OutputMsg("Logon failed!");
```

Running Script Files in the Development Environment

If a script file is being run from the SDK, sufficient time should be allowed for the screen to update between commands.

For example, if the script contains the following code

```
for (loop = 0; loop < 10; loop++)
{
    NSW.SendKey(-1, -1, PF3);
```

you should add a `Pause` statement to allow for the extra screen processing required in the development version:

```
for (loop = 0; loop < 10; loop++)
{
    NSW.SendKey(-1, -1, PF3);
    NSW.Pause(1500);
```

The NSW Object

Entire Screen Builder adds an object named NSW to the core JavaScript language. This object allows to deal with host sessions in the script by offering several methods and properties.

This chapter provides the following general information:

- Syntax Conventions
 - NSW Object Properties
 - NSW Object Methods
 - User Variables
 - Error Codes
 - Display Status
 - Windows Message Box Flags
 - Windows Button Identifiers
-

Syntax Conventions

The following syntax conventions are used in the method descriptions:

Convention	Description
monospace font	Items in monospace font are keywords (for example, methods and properties). You must specify these items as indicated in the syntax.
<i>italics</i>	Items in italics are placeholders for information that you must provide. To signal case-sensitivity or embedded blanks, you must enclose the item in double or single quotation marks (" " or ' ').
[]	Square brackets indicate that the enclosed item is optional.
{ }	Curly braces indicate that one of the enclosed items is required.
	Vertical lines are used as separators between alternative parameter values.
...	Ellipsis are used when the item preceding the ellipsis can be repeated.

NSW Object Properties

Property	Description
<code>errno</code>	The <code>errno</code> property contains the error code of the last error that occurred. See Error Codes for a list of all defined values. To reset the <code>errno</code> property, set it to 0.
<code>isConnected</code>	True if there is a session for the client, otherwise false.
<code>maxRows</code>	Maximum number of lines on the character screen. Depends on the host session properties. Currently set to 24.
<code>maxColumns</code>	Maximum number columns per line on the character screen. Depends on the host session properties. Currently set to 80.

NSW Object Methods

The methods of the NSW object can be grouped into the following categories:

- Host Communication Methods
- User Interface Methods
- Processing Methods
- Client-Side Access Methods

See the corresponding sections later in this documentation for detailed descriptions of all methods.

The methods use different ways for returning information to the caller:

- return value, and
- `errno` property.

Some methods do not have a return value. Some methods (for example, `GetUserInput`) also return information in `NSW(errno)`.

In case of an error condition, the methods set `NSW(errno)` to a method-specific error code. `NSW(errno)` is set to `NSW_NO_ERROR(0)` if there was no error.

Runtime errors in the NSW methods return an error code in `NSW(errno)` if the error is handled in the method.

The script aborts in the following cases when a statement containing an error is detected:

- syntax error in a core JavaScript statement,
- syntax error in an NSW method,
- runtime error in a core JavaScript statement,
- unhandled runtime error in an NSW method.

User Variables

User variables are persistent data that are stored in the user's profile in the Entire Screen Builder Server. This allows to have user-defined parameters (such as passwords for host logon) which can be used in script files without having to change the script when you have to change the parameter value.

A user variable consists of two parts: variable name and value. The user variable values are encrypted before they are stored in the user profile.

You create and set a user variable with the NSW method `SetUserVariableValue` in a script file. You can use the same method to change the value of an existing user variable.

While working with a viewer, it is also possible to change the values of your defined user variables in a dialog box. See *Defining User Variables* in the *Utilities* documentation.

To retrieve user variables from your user profile, use the NSW method `GetUserVariableValue`.

To delete a user variable, use the NSW method `DeleteUserVariable`.

Error Codes

The `errno` property contains the error code of the last NSW method that was executed. The following table lists all defined values.

Error Code	Keyword	Description
0	NSW_NO_ERROR	Success.
1	NSW_SESSION_OPEN	A session has already been opened.
2	NSW_SESSION_ERROR	An error has occurred in the session (depends on the method).
3	NSW_RANGE_ERROR	Cursor position error.
4	NSW_DATA_SYNTAX_ERROR	File name specification error.
5	NSW_UNKNOWN_ERROR	Unspecified error.
6	NSW_ACCESS_DENIED_ERROR	Access has been denied.
7	NSW_HANDLE_ERROR	The handle is invalid.
8	NSW_FILE_NOTFOUND_ERROR	The file has not been found.

Error Code	Keyword	Description
9	NSW_PATH_NOTFOUND_ERROR	The path has not been found.
10	NSW_INVALID_NAME_ERROR	Invalid syntax for volume, path or file name.
11	NSW_DIR_NOTEEMPTY_ERROR	The directory is not empty.
12	NSW_SHARING_VIOLATION_ERROR	The process cannot access the file because it is being used by another process.
13	NSW_DISK_FULL_ERROR	The disk is full.
14	NSW_NOT_READY_ERROR	The device is not ready.
15	NSW_ALREADY_EXISTS_ERROR	The file already exists.
16	NSW_WRITE_PROTECTED_ERROR	The medium is write-protected.
17	NSW_NEGATIVE_SEEK_ERROR	An attempt was made to move the file pointer before the beginning of the file.
18	NSW_TIMEOUT_ERROR	A timeout occurred.
19	NSW_BAD_DLL_ERROR	No valid DLL.
20	NSW_DLL_NOT_FOUND_ERROR	The DLL has not been found.
21	NSW_FUNC_NOT_FOUND_ERROR	The function has not been found in the DLL.
22	NSW_USERVALUE_NOTFOUND_ERROR	The user value has not been found.
23	NSW_USERVALUE_NOTSETTABLE_ERROR	The user value cannot be set.
24	NSW_NO_SESSION_SPECIFIED	A session has not been specified.
25	PROC_ERROR_NO_OPEN_SESSION	A session has not been opened.
26	PROC_USER_VALUE_NOT_GETABLE	Access to this user variable is not allowed.
27	PROC_USER_ACTION_NOT_PERMITTED	The requested action is not allowed for this user variable.
28	PROC_USER_HAS_NO_DEFINED_SESSION	The current user has no defined default session.
29	PROC_USER_ACTION_CANCELED	The user has chosen the Cancel button in the Logon dialog box.
30	PROC_ACTION_NOT_SUPPORTED	This action is not supported by the current viewer.
31	PROC_ARGUMENT_NOT_FOUND	The user argument has not been defined.
32	PROC_ERROR_NO_FILE_RETURNED	The user has chosen the Cancel button in the Browse File dialog box.

Display Status

Display Status	Keyword	Description
1	DISPLAY_ON	The screen is updated.
2	DISPLAY_OFF	The screen is not updated.

Windows Message Box Flags

Flag	Meaning
MB_OK	The message box contains one push button: OK . This is the default.
MB_OKCANCEL	The message box contains two push buttons: OK and Cancel .
MB_RETRYCANCEL	The message box contains two push buttons: Retry and Cancel .
MB_YESNO	The message box contains two push buttons: Yes and No .
MB_YESNOCANCEL	The message box contains three push buttons: Yes , No and Cancel .

Windows Button Identifiers

Identifier	Meaning	Value
IDCANCEL	The Cancel button was selected.	2
IDNO	The No button was selected.	7
IDOK	The OK button was selected.	1
IDRETRY	The Retry button was selected.	4
IDYES	The Yes button was selected.	6

Host Communication Methods

The following host communication methods are available:

- CloseSession
 - GetDefaultSession
 - GetDisplay
 - GetScreenText
 - OpenSession
 - OpenUpdatedSession
 - SendKey
 - SetDataTransfer
 - SetDisplay
 - SetSessionParameter
 - SetUnixLogonCredentials
 - SetXMLEncoding
 - SetXMLStyleSheet
 - TypeString
 - UseDefinedSession
-

CloseSession

Close the current session.

Syntax

```
CloseSession()
```

Parameters

None.

Description

The `CloseSession` method closes the current host session.

If you want to close a session with the `CloseSession` method and after this open another session with `OpenSession()` or `OpenUpdatedSession()`, you should make sure that there is a delay of at least 2 seconds between these two statements. This delay is required because `CloseSession()` does not synchronize. A delay can be defined, for example, with the statement `NSW.Pause (2000)`.

Example

```
NSW.CloseSession();
```

Return Values

None.

GetDefaultSession

Get the default session.

Syntax

```
GetDefaultSession()
```

Parameters

None.

Description

The `GetDefaultSession` method returns the name of the defined default session for the current user. This value can then be used to open a session using the methods `UseDefinedSession` or `OpenSession`.

Example

```
szSession = NSW.GetDefaultSession();
```

Return Values

Value of the variable. If the variable was not found, an empty string is returned and `errno` is set to `PROC_USER_HAS_NO_DEFINED_SESSION`.

GetDisplay

Return the current state of the display flag.

Syntax

```
GetDisplay( )
```

Parameters

None.

Description

The `GetDisplay` method returns the current state of the display flag which has been set with the `SetDisplay` method.

Example

```
iDispValue = NSW.GetDisplay();
```

Return Values

Either `DISPLAY_ON` or `DISPLAY_OFF`.

GetScreenText

Get text from the terminal emulation screen.

Syntax

```
GetScreenText(StartRowPos, StartColPos, Length)
```

Parameters

StartRowPos Values from 1 through NSW.maxRows.

StartColPos Values from 1 through NSW.maxColumns.

Length Values from 1 through NSW.maxRows * NSW.maxColumns.

Description

The GetScreenText method gets the text from the current terminal emulation screen starting in *StartRowPos* and *StartColPos*. The length of this text is determined by the *Length* parameter.

Example

Get the contents of the complete terminal emulation screen:

```
strValue = NSW.GetScreenText(1,1,NSW.maxRows*NSW.maxColumns);
```

Return Values

A string.

OpenSession

Open a session.

Syntax

```
OpenSession(Name[, Keyword, Value]...)
```

Parameters

Name A valid session name.

Keyword The parameter to be changed.

Value The new value for the parameter.

Description

The OpenSession method opens the host session with the specified session name.

You can specify more than one value pair consisting of *Keyword* and *Value*.

Optionally the session parameters can be changed. See the description of the SetSessionParameter method for a list of all valid parameters.

Examples

```
Result = NSW.OpenSession("Darmstadt Host");
Result = NSW.OpenSession("Darmstadt Host", HOST_TCPIP_PORT, 23);
Result = NSW.OpenSession("Darmstadt Host", HOST_TCPIP_PORT, 23, HOST_TN3270E, true);
```

Return Values

True if the method was successful. False if there was an error.

OpenUpdatedSession

Open the changed session.

Syntax

```
OpenUpdatedSession()
```

Parameters

None.

Description

The `OpenUpdatedSession` method opens the host session which was defined with a previous call to the method `UseDefinedSession` and where the parameters have been changed with the method `SetSessionParameter`.

This method only makes sense when being used together with the methods `UseDefinedSession` and `SessionParameter`.

Example

```
NSW.OpenUpdatedSession();
```

Return Values

True if the method was successful. False if there was an error.

If `UseDefinedSession` has not been called previously, this method returns false and `errno` is set to `NSW_NO_SESSION_SPECIFIED`.

SendKey

Send a terminal emulation key to the host session.

Syntax

```
SendKey(Row, Column, Keycode)
```

Parameters

<i>Row</i>	The number of the row to which the cursor is to be moved. Or -1 for the current position of the terminal emulation cursor.
<i>Column</i>	The number of the column to which the cursor is to be moved. Or -1 for the current position of the terminal emulation cursor.
<i>Keycode</i>	A terminal emulation key. See <i>Terminal Emulation Keys</i> in Entire Screen Builder's <i>System Management Hub</i> documentation.

Description

The *SendKey* method sends the specified terminal emulation key to the current host session. If both *Row* and *Column* are positive values, the cursor is moved to the specified position first. The *SendKey* method then waits for a response from the host before processing continues with the next instruction.

To send PF keys in AS/400 style, the session has to be defined as an AS/400 session. See *Communication Properties for Telnet TN3270* in Entire Screen Builder's *System Management Hub* documentation.

Examples

```
Result = NSW.SendKey(-1, -1, PF3);
Result = NSW.SendKey(-1, -1, AS_PF3);
```

Return Values

True or false. In case of an error, *errno* is set to 1.

SetDataTransfer

Set the PC file name for Natural data transfer.

Syntax

```
SetDataTransfer(SetCommand)
```

Parameters

SetCommand A valid SET command for data transfer:

```
SET PCFILE x DOWN DATA filename.ext
SET PCFILE x UP DATA filename.ext
SET PCFILE y DOWN REPORT filename.ext
```

Description

With the SetDataTransfer method, you can set the file name for subsequent uploads or downloads. *x* is a valid work file number. *y* is a valid printer file number. *filename.ext* is a file on the client workstation.

To reset the number for a work file or printer file, use the SET command without specifying a file name.

See also: *Specifying a File Name Using the SET Command* in the *Data Transfer* documentation.

Example

```
Result = NSW.SetDataTransfer("SET PCFILE 7 DOWN DATA C:\\TEMP\\TEST.NCD");
```

Return Values

True if the method was successful. False if there was an error.

SetDisplay

Stop the screen being updated in the client.

Syntax

```
SetDisplay(Flag)
```

Parameters

Flag Either DISPLAY_OFF or DISPLAY_ON.

Description

DISPLAY_OFF stops the screen being updated in the client until the next call to DISPLAY_ON.

If the script ends in DISPLAY_OFF mode, the screen is automatically refreshed.

Example

```
NSW.SetDisplay(DISPLAY_OFF);
```

Return Values

True if the method was successful. False if there was an error.

SetSessionParameter

Change the parameters of the selected session temporarily.

Syntax

```
SetSessionParameter(Keyword, Value[, Keyword, Value]...)
```

Parameters

Keyword The parameter to be changed. See the table below for a list of valid keywords.

Value The new value for the parameter.

Description

With the `SetSessionParameter` method, you can change the parameters of the session that has previously been selected with the `UseDefinedSession` method. The changed session can then be opened by calling the method `OpenUpdatedSession`.

You can specify more than one value pair consisting of *Keyword* and *Value*.

Important:

The parameters changed with `SetSessionParameter` are only changed inside the JavaScript code. Changing session parameters here has no effect on the "real" session as defined with the System Management Hub.

You can redefine the session temporarily, without having to change the settings in the System Management Hub and without having to restart the server. For example, you can prompt for the input of a user name and a password and then use the obtained values for the session by assigning them to `HOST_USER_ID` and `HOST_PASSWORD`.

The following table lists all valid keywords and indicates the appropriate type(s) of host session.

Keyword	Type	Description	Telnet TN3270	Telnet VT	BS2000	Natural UNIX
HOST_TCPIP_PORT	Integer	TCP/IP port.	X	X	X	X
HOST_TCPIP_ADDR	String	TCP/IP address.	X	X	X	X
HOST_TERMINAL_TYPE	String	Terminal type.	X Valid values: IBM-3278-2 IBM-3278-4 IBM-3278-2-E IBM-3278-4-E IBM-3279-2 IBM-3279-4 IBM-3279-2-E IBM-3279-4-E IBM-3278-3 IBM-3278-5 IBM-3278-3-E IBM-3278-5-E IBM-3279-3 IBM-3279-5 IBM-3279-3-E IBM-3279-5-E	X Valid values: VT100 VT220 VT320		
HOST_IGNORE_EABS	Boolean	Ignore extended attribute bytes in data stream.	X			
HOST_TN3270E	Boolean	Open the session as a TN3270E session.	X			
HOST_DEVICE_NAME	String	Use device name for a TN3270E session.	X			
HOST_EOS_DELAY	Integer	End-of-screen delay.	X	X	X	
HOST_USER_ID	String	User name.				X
HOST_PASSWORD	String	Password (for user).				X
HOST_SHOW_NILS	Boolean	Display NIL characters (dots) in BS2000.			X	
HOST_BS2000_APPLIC	String	BS2000 application.			X	
HOST_STATION_NAME	String	BS2000 station name.			X	
HOST_COMPRESSED	Boolean	Compressed session.				X
HOST_LOGON_CREDENTIALS	Boolean	Logon credentials.				X
HOST_VT_LINE	Integer	Line size: 80 or 132.		X		

Keyword	Type	Description	Telnet TN3270	Telnet VT	BS2000	Natural UNIX
HOST_VT_CR_OPTION	Constant	Return key send option		X Valid values: VT_CR_OPTION_PLAIN_CR (Plain CR) VT_CR_OPTION_CR_LF (CR + LF) VT_CR_OPTION_CR_NULL (CR + NULL)		

Examples

```
Result = NSW.SetSessionParameter(HOST_TERMINAL_TYPE, "IBM-3279-2-E");
Result = NSW.SetSessionParameter(HOST_TCPIP_PORT, 23, HOST_TN3270E, true);
```

Return Values

True if the method was successful. False if there was an error.

If `UseDefinedSession` has not been called previously, this method returns false and `errno` is set to `NSW_NO_SESSION_SPECIFIED`.

SetUnixLogonCredentials

Terminal Viewer only.

Force a UNIX logon before a session can be opened.

Syntax

```
SetUnixLogonCredentials()
```

Parameters

None.

Description

Use this method before an OpenSession call.

If a JavaScript method is opening a Natural UNIX session that requires user logon credentials, SetUnixLogonCredentials must be called first. Otherwise, the dialog prompting for user name and password is not shown.

Example

```
bUserClickedOK = NSW.SetUnixLogonCredentials();
```

Return Values

True if the user has chosen the **OK** button. False if the user has not chosen the **OK** button or if the function is not supported.

SetXMLEncoding

Set encoding for data transfer with HTML and XML.

Syntax

```
NSW.SetXMLEncoding (Encoding)
```

Parameters

Encoding The XML/HTML encoding.

Description

The SetXMLEncoding method defines the encoding that is to be used for data transfer with HTML and XML. When a connection has been established with a viewer, this method temporarily overwrites the setting made in the session definition. When the viewer is disconnected, the encoding from the session definition is used again. See *General Properties* in the *Host Sessions* section of Entire Screen Builder's *System Management Hub* documentation.

See also the following sections in the *Data Transfer* documentation:

- *Downloading Data to HTML*
- *Downloading Data to XML*
- *Uploading XML Files*

The following example shows the header for an XML file that results from executing SetXMLEncoding ("utf-8"):

```
<?xml version="1.0" encoding="utf-8"?>
```

To cancel the usage of the encoding, you have to call SetXMLEncoding with an empty string parameter. In this case, the above XML header is written with the following default encoding:

```
<?xml version="1.0" encoding="windows-1252"?>
```

Examples

```
// Add encoding definition  
NSW.SetXMLEncoding ("utf-8")  
  
// Reset to default encoding  
NSW.SetXMLEncoding ("")
```

Return Values

None.

SetXMLStyleSheet

Define style sheet for XML download.

Syntax

```
NSW.SetXMLStyleSheet (Type, HRef)
```

Parameters

Type The type of XML style sheet.

HRef The location of the style sheet.

Description

The SetXMLStyleSheet method defines the style sheet that is to be used when downloading data to an XML file (see *Downloading Data to XML* in the *Data Transfer* documentation). After executing this method, all downloaded XML files contain a style sheet definition like the following:

```
<?xml:stylesheet type="text/xsl" href="http://pcnha/xml/employ2.xsl"?>
```

To cancel the usage of the style sheet, you have to call SetXMLStyleSheet with empty string parameters. In this case, the style sheet definition is no longer written to the downloaded XML files.

Note:

It is also possible to specify a style sheet using the SET command. The last command that was issued (either SetXMLStyleSheet or SET) always determines the style sheet to be used.

Examples

```
// Insert style sheet definition
NSW.SetXMLStyleSheet ("text/xsl", "http://pcnha/xml/employ2.xsl")

// Cancel usage of style sheet
NSW.SetXMLStyleSheet ("", "")
```

Return Values

None.

TypeString

Send a string or a string plus a terminal emulation function key to the current host session.

Syntax

```
TypeString(Row, Column, String, [Keycode])
```

Parameters

<i>Row</i>	A number value. Valid values are 1 through NSW.maxRows, or -1.
<i>Column</i>	A number value. Valid values are 1 though NSW.maxColumns, or -1.
<i>String</i>	A string value.
<i>Keycode</i>	A terminal emulation key. See <i>Terminal Emulation Keys</i> in Entire Screen Builder's <i>System Management Hub</i> documentation.

Description

The string is typed into the host session at the specified cursor position.

If both values for *Row* and *Column* are -1, the string is typed at the current cursor position and the cursor is moved. For sessions of type Natural UNIX, this does not work in all situations. It is recommended that you always use the exact cursor position when working with this type of session.

If a valid cursor position is given in *Row* and *Column*, the string is typed at this position and the cursor is moved. Note that the cursor must be in an unprotected (input) field to allow typing in.

For sessions of type Natural UNIX, this call only makes sense, if *Keycode* is provided. Otherwise, "CR" is sent.

Examples

```
NSW.TypeString(-1, -1, "MyUserID");
NSW.TypeString(-1, -1, "MyUserID", CR);
NSW.TypeString(2, 11, "abcd");
```

Return Values

True or false. In case of an error, errno is set.

UseDefinedSession

Select a session.

Syntax

```
UseDefinedSession(Name)
```

Parameters

Name A valid session name.

Description

The UseDefinedSession method selects the session to be used with the methods SetSessionParameter and OpenUpdatedSession.

This call only makes sense when called together with SetSessionParameter and OpenUpdatedSession.

Example

```
Result = NSW.UseDefinedSession("Darmstadt Host");
```

Return Values

True if the method was successful. False if there was an error.

User Interface Methods

The following user interface methods are available:

- Cancel
 - DeleteUserVariable
 - GetClientArgument
 - GetUserConnectionValue
 - GetUserInput
 - GetUserVariableValue
 - MessageBox
 - OutputMsg
 - SetClientArgument
 - SetUserConnectionValue
 - SetUserVariableValue
-

Cancel

Cancel processing of the script file immediately.

Syntax

```
Cancel()
```

Parameters

None.

Description

The Cancel method stops processing and terminates the script. This can also be called from a nested function.

Example

```
NSW.Cancel();
```

Return Values

None.

DeleteUserVariable

Delete a user-defined variable.

Syntax

```
DeleteUserVariable (Variable)
```

Parameters

Variable A user-defined variable (string).

Description

The DeleteUserVariable method removes a user-defined variable that has been set with the SetUserVariableValue method.

Example

```
bRet = NSW.DeleteUserVariable ("myvar");
```

Return Values

If the method is successful, it returns true.

GetClientArgument

Return values defined by a client-side user exit (API).

Syntax

```
GetClientArgument(Name)
```

Parameters

Name Name of a script argument defined in the client.

Description

The GetClientArgument method returns the values defined by the user exit (API) method SetScriptArgument.

String comparison is case-sensitive.

Example

```
szArgValue = NSW.GetClientArgument( "param1" );
```

Return Values

The defined value for this argument.

If the argument is not set, an empty string is returned and errno is set to PROC_ARGUMENT_NOT_FOUND.

GetUserConnectionValue

Return the value of a user-specific connection variable.

Syntax

```
 GetUserConnectionValue (Variable)
```

Parameters

Variable A user-specific connection variable. See the table below.

Description

The `GetUserConnectionValue` method returns the value of a user-specific connection variable.

One of the following user connection variables has to be specified for the *Variable* parameter:

Variable	Description
NSW_CLIENT_TCP_IP_ADDR	IP address of the client.
NSW_CLIENT_HOST_NAME	Host name of the client.
NSW_USER_NAME	User name.

Thus, if you want to know the name of the current user, you have to specify `NSW_USER_NAME` for the *Variable* parameter.

Example

```
szValue = NSW.GetUserConnectionValue (NSW_CLIENT_TCP_IP_ADDR);
```

Return Values

Value of the variable.

If the variable was not found, an empty string is returned and `errno` is set to `NSW_USERVALUE_NOTFOUND_ERROR`.

GetUserInput

Display a dialog box in which the user can enter text.

Syntax

```
 GetUserInput(Prompt1, Prompt2, Flags, InitialValue[, Length])
```

Parameters

<i>Prompt1</i>	A string that is displayed in the dialog box.
<i>Prompt2</i>	A string that is displayed in a second line in the dialog box.
<i>Flags</i>	Can be a combination of the following: REQUIRED_DATA, LENGTH_LIMIT, STRING_DATA, NUMERIC_DATA and PASSWORD_CHAR.
<i>InitialValue</i>	A default text that is displayed in an input field of the dialog box.
<i>Length</i>	Maximum length of the input field. If LENGTH_LIMIT is set, <i>Length</i> must also be valid.

Description

The `GetUserInput` method displays a dialog box in the viewer of the client workstation. It returns the data entered by the user and an identifier for the button pressed by the user.

Example

```
strPassword = NSW.GetUserInput("Password", "Enter your password", REQUIRED_DATA | LENGTH_LIMIT | PASSWORD_CHAR, 12);
strFromUser = NSW.GetUserInput("Enter a number", "", NUMERIC_DATA);
```

Return Values

The data entered by the user.

The property `NSW(errno)` will be set to 0, if the user chooses the **OK** button. It will be 1, if the user chooses the **Cancel** button.

GetUserVariableValue

Return the value of a user-defined variable.

Syntax

```
 GetUserVariableValue ( Variable )
```

Parameters

Variable A user-defined variable (string).

Description

The GetUserVariableValue method returns the value of a user-defined variable that has been set with the SetUserVariableValue method.

Example

```
szValue = NSW.GetUserVariableValue ( "myvar" );
```

Return Values

Value of the variable.

If the variable was not found, an empty string is returned and errno is set to NSW_USERVALUE_NOTFOUND_ERROR.

MessageBox

Pause processing of the script file and display a message box.

Syntax

```
MessageBox ( Text[ , Flags] )
```

Parameters

Text String to be shown.

Flags A combination of the flags for a Windows message box (for example, MB_OK or MB_YESNO).

Description

The MethodBox method displays a message box in the viewer on the client workstation. Processing of the script file is resumed when the user chooses a command button in the message box.

Example

```
NSW.MessageBox( "Error opening session" );
ReturnCode = NSW.MessageBox( "Good morning" , MB_OK );
```

Return Values

ReturnCode is set to the Windows identifier of the button that has been chosen (for example, IDOK or IDCANCEL).

OutputMsg

Send non-blocking output to the viewer.

Syntax

```
OutputMsg(String)
```

Parameters

String String to be shown on the client side.

Description

The OutputMsg method sends a message text to the viewer.

For the GUI viewers, this output appears in the status line.

For the Terminal Viewer, this output appears in the output window.

Example

```
NSW.OutputMsg( "Now reading database" );
```

Return Values

None.

SetClientArgument

Define arguments to be processed by a client-side user exit (API).

Syntax

```
SetClientArgument(Name)
SetClientArgument(Name, Value)
```

Parameters

Name Name of a script argument.

Value Value for this argument.

Description

The SetClientArgument method defines the arguments to be returned to the client and to be processed by the user exit (API) method GetScriptArgument.

If only the *Name* is given, the argument is deleted from the list of client arguments.

Example

```
bArgDel = NSW.SetClientArgument("param1");
bArgSet = NSW.SetClientArgument("param1", "retval");
```

Return Values

If the method is successful, it returns true.

If the argument is not set or if it has been deleted, false is returned and errno is set to PROC_ARGUMENT_NOT_FOUND.

SetUserConnectionValue

Set the value of a user-specific connection variable.

Syntax

```
SetUserConnectionValue (Variable, Value)
```

Parameters

Variable A defined user-specific variable.

Value Value to which the variable is to be set.

Description

The SetUserConnectionValue method sets the value of a user-specific variable.

The following user connection variable has to be specified for the *Variable* parameter:

Variable	Description
NSW_USER_PASSWORD	User password.

Example

```
bRet = NSW.SetUserConnectionValue (NSW_USER_PASSWORD, "abc");
```

Return Values

If the method is successful, it returns true.

If the variable was not found, false is returned and `errno` is set to `NSW_USERVALUE_NOTFOUND_ERROR`.

If you try to set a variable which cannot be modified (such as `NSW_CLIENT_HOST_NAME` which is only allowed with the `GetUserConnectionValue` method), `errno` is set to `NSW_USERVALUE_NOTSETTABLE_ERROR`.

SetUserVariableValue

Set the value of a user-defined variable.

Syntax

```
SetUserVariableValue (Variable, Value)
```

Parameters

Variable A user-defined variable (string).

Value Value to which the variable is to be set.

Description

The SetUserVariableValue method defines a variable and sets its value.

For example, you can define a variable with the name "myvar" and set its value to "abc". This variable is then permanently stored for the current user and can be retrieved (for example, after a restart of the server) with GetUserVariableValue("myvar").

Example

```
bRet = NSW.SetUserVariableValue ("myvar", "abc");
```

Return Values

If the method is successful, it returns true.

Processing Methods

The following processing methods are available:

- CallExternalFunction
 - Pause
-

CallExternalFunction

Call a DLL function.

Syntax

```
CallExternalFunction (DLLname, Functionname[ , Parameterlist])
```

Parameters

<i>DLLname</i>	A valid DLL name, including the path.
<i>Functionname</i>	Name of the function in the DLL.
<i>Parameterlist</i>	One or more parameters to be used for the function.

Description

The `CallExternalFunction` method calls the specified function in the DLL. A callback pointer and the values from the *Parameterlist* are passed to the function as explained below.

The function prototype for the functions in the external library is:

```
int functionname (PSAGJSCALLBACK pCallback, int argc, char** argv)
```

The first parameter (`PSAGJSCALLBACK`) is a pointer to the following structure:

```
typedef struct tagSAGJSCALLBACK
{
    CALLBACK_PROC SetJSVariable;
    CALLBACK_PROC GetJSVariable;
}SAGJSCALLBACK, *PSAGJSCALLBACK;
```

Two callback procedures are provided that enable you to communicate with the JavaScript that called the function: `GetJSVariable` and `SetJSVariable`. You use these functions to access variables in the JavaScript by name to retrieve or set the values of these variables. The procedures have the following syntax:

```
SetJSVariable (char* szVarName, char* szVarValue)
GetJSVariable (char* szVarName, char* szVarValue)
```

The second parameter (`int argc`) passes the number of the parameters that are in the *Parameterlist*.

The third parameter (`char** argv`) is a pointer to an array of strings that contains the values from the *Parameterlist*. Note that values are passed as strings.

Example

```
iFunctionRet= NSW.CallExternalFunction ('f:/mylib/mylib.dll', 'fnMylib', '10', '4', 'x')
```

When the function `fnMylib` is called, the second parameter (`int argc`) has the value 3 since there are three values in the *Parameterlist*. The third parameter (`char** argv`) points to a string array with the three strings "10", "4" and "x" (in that order) in the array.

The following is an example JavaScript program which calls the function `fnAdd2` in *mylib.dll*. The function `fnAdd2` uses the callback functions to retrieve the value of the script variable named `i`, to add 2 to it and to put the new value to the script variable.

```
i=1;
NSW.CallExternalFunction ('f:/mylib/mylib.dll', 'fnAdd2')
NSW.MessageBox ("i = " + i)
```

This is the function `fnAdd2` in the DLL:

```
int fnAdd2(PSAGJSCALLBACK pCallback, int argc, char** argv)
{
    char* szValue = new char(20);

    // Get the variable 'i' from JavaScript
    pCallback->GetJSVariable("i", szValue);

    // convert value to integer
    int iVal = atoi(szValue);
    iVal += 2;

    // convert value back to string
    itoa (iVal, szValue, 10);

    // Set the variable 'i' in JavaScript
    pCallback->SetJSVariable("i", szValue);

    return 1;
}
```

Return Values

Return value of the called DLL function. This must be an `int` value.

Pause

Pause processing of the script file.

Syntax

```
Pause(Number)
```

Parameters

Number The number of milliseconds that the script is to be paused.

Description

Pauses the execution of the script for the specified number of milliseconds.

Example

```
NSW.Pause(1000);
```

Return Values

None.

Client-Side Access Methods

Client-side access methods allow to access resources (i.e. files) on the client workstation where the viewer runs from a script file that is executed on the Entire Screen Builder Server.

When one of these methods fails, it returns an error code in `NSW(errno)`. Otherwise, `NSW(errno)` is set to `NSW_NO_ERROR (0)`. See *Error Codes* for a list of all error codes.

The following client-side access methods are available:

- `BrowseClientFile`
 - `CloseClientFile`
 - `CreateClientFolder`
 - `DeleteClientFile`
 - `DeleteClientFolder`
 - `ExecuteClientApplication`
 - `ListClientDirectory`
 - `OpenClientFile`
 - `ReadClientFile`
 - `SetClientFilePtr`
 - `SetCurrentClientFolder`
 - `WriteClientFile`
-

BrowseClientFile

Display an Open File dialog box on the client.

Syntax

```
BrowseClientFile(Filter, Name)
```

Parameters

Filter Defines the filter to be used for the dialog. If this parameter is not set, the following default entry is used:

```
All Files (*.*)|*.|| |
```

Example for Microsoft Excel:

```
szFilter = "Chart Files (*.xlc)|*.xlc|Worksheet Files (*.xls)|*.xls|  
Data Files (*.xlc;*.xls)|*.xlc; *.xls|All Files (*.*)|*.|| |"
```

For a full description of the filter string, go to <http://msdn.microsoft.com/> and search for "CFileDialog::CFileDialog".

Name Any default name to be displayed in the Open File dialog box. If this parameter is not set, the file name is initially empty.

Description

The `BrowseClientFile` method displays an Open File dialog box. A filter can be defined for the file type, and the name of the file to be opened can be defined.

Examples

```
szFileName = NSW.BrowseClientFile();  
szFileName = NSW.BrowseClientFile("", "test.txt");  
szFileName = NSW.BrowseClientFile("Text Files (*.txt)|*.txt|All Files (*.*)|*.|| |");  
szFileName = NSW.BrowseClientFile("Ini Files (*.ini)|*.ini|All Files (*.*)|*.|| |", "test.ini");
```

Return Values

The file name selected by the user. This can then be used in other file access functions.

If the user chooses the **Cancel** button, `errno` is set to `PROC_ERROR_NO_FILE_RETURNED` and the file name is set to blank.

CloseClientFile

Close a file on the client.

Syntax

```
CloseClientFile(Filehandle)
```

Parameters

Filehandle File handle returned by a previous `OpenClientFile()` call.

Description

The `CloseClientFile` method closes the file that is identified by the given file handle. After this method call, the file handle is no longer valid.

Example

```
bReturn = NSW.CloseClientFile (hFile);
```

Return Values

True if the method was successful. False if there was an error.

CreateClientFolder

Create a folder on the client.

Syntax

```
CreateClientFolder (Foldername)
```

Parameters

Foldername The name for the new folder.

Description

The `CreateClientFolder` method creates a new folder. Its name is specified with the parameter *Foldername*.

When an absolute path has been specified with the parameter *Foldername* (for example, `c:/test`), the folder is created at the specified location. When a relative path has been specified (for example, `my_new_folder`), the folder is created in the current folder.

If this method fails (e.g. no write access on the drive or the folder already exists), the method returns false and the `errno` property indicates the reason for the failure.

Example

```
bReturn = NSW.CreateClientFolder ("C:/mydir");
```

Return Values

True if the method was successful. False if there was an error.

DeleteClientFile

Delete a file on the client.

Syntax

```
DeleteClientFile (Filename)
```

Parameters

Filenname The name of the file to be deleted.

Description

The DeleteClientFile method removes the specified file from the disk.

Example

```
bReturn = NSW.DeleteClientFile ("C:/mydir/test.txt");
```

Return Values

True if the method was successful. False if there was an error.

DeleteClientFolder

Delete a folder on the client.

Syntax

```
DeleteClientFolder (Foldername)
```

Parameters

Foldername The name of the folder to be deleted.

Description

The DeleteClientFolder method removes the specified folder.

Example

```
bReturn = NSW.DeleteClientFolder ("C:/mydir");
```

Return Values

True if the method was successful. False if there was an error.

ExecuteClientApplication

Execute an application on the client.

Syntax

When blocking is used:

```
ExecuteClientApplication (Applicationname, Applicationparameters, Blocking[, Blockingtimeout])
```

When blocking is not used:

```
ExecuteClientApplication (Applicationname[, Applicationparameters])
```

Parameters

<i>Applicationname</i>	The name of the application that is to be started on the client. May be an empty string.
<i>Applicationparameters</i>	Parameters to be passed to the application when it is started (String value).
<i>Blocking</i>	Boolean value. Default: false.
<i>Blockingtimeout</i>	Numeric value. Only used if <i>Blocking</i> is set to true. The number of milliseconds the method is to wait before it returns. If not set, the method waits infinitely.

Description

The `ExecuteClientApplication` method executes the specified application on the client. With the second method parameter, you can pass parameters to the specified application.

If *Blocking* is set to true, the method waits until the application has finished before it returns. If *Blocking* is set to false, it returns immediately.

The fourth parameter *Blockingtimeout* is only valid if *Blocking* is active. You can specify a timeout value. If *Blocking* is active and no timeout value was specified, the method never times out.

Example

```
bRet = NSW.ExecuteClientApplication ("excel", "c:\\program files\\test1.xls", true, 50000)
```

Return Values

True if the method was successful. False if there was an error.

If a timeout occurs, the method returns false and the property `NSW(errno)` will be set to `NSW_TIMEOUT_ERROR`.

ListClientDirectory

Return all folders and files in a specific folder of the client.

Syntax

```
ListClientDirectory (Foldername)
```

Parameters

Foldername A valid folder name or an asterisk (*).

Description

The `ListClientDirectory` method returns the folders and files in the specified folder. The list of folders and files is returned as a string value. The items in the list are separated by newline ("\\n") characters. For a folder, the string "<dir>" is added to its name. For example: "temp <dir>".

You can also call `ListClientDirectory` with an asterisk (*) as the string argument. In this case, the content of the current folder is listed.

Example

```
SzFileList = NSW.ListClientDirectory ("C:/temp/*");
```

Return Values

A string containing all files in the specified folder.

OpenClientFile

Open a file on the client.

Syntax

```
OpenClientFile (Filename, Accessflag)
```

Parameters

Filenname The name of the file to be opened.

Accessflag Can be one of the following:
NSW_READWRITE for read/write access.
NSW_READONLY for read-only access.

Description

The OpenClientFile method opens the specified file (either with read/write access or read-only access, according to the setting of the second parameter) and returns a file handle.

Example

```
hFile = NSW.OpenClientFile ("C:/mydir/test.txt", NSW_READWRITE);
```

Return Values

The method returns a file handle. In case of an error, -1 is returned and the error code is set in NSW(errno).

ReadClientFile

Read the contents of a file on the client.

Syntax

```
ReadClientFile (Filehandle)
```

Parameters

Filehandle File handle returned by a previous `OpenClientFile()` call.

Description

The `ReadClientFile` method reads the contents of the file that is identified by the given file handle. It starts at the position of the current file pointer and ends at the end of the file. The text from the file is returned as a string.

Example

```
szReturn = NSW.ReadClientFile (hFile);
```

Return Values

A string containing the text from the file. If an error occurs, an empty string is returned and the error code is set in `NSW(errno)`.

SetClientFilePtr

Set a pointer in a file on the client.

Syntax

```
SetClientFilePtr (Filehandle, Positionflag[, Offset])
```

Parameters

Filehandle File handle returned by a previous `OpenClientFile()` call.

Positionflag Can be one of the following:

`NSW_END_OF_FILE` to point to the end of the file.

`NSW_START_OF_FILE` to point to the beginning of the file.

`NSW_CURR_FILE_POS` to point to the current position in the file.

Offset Optional numeric parameter which may also be negative. Indicates the offset to the position flag in bytes.

Description

The `SetClientFilePtr` method sets the file pointer position in the file that is identified by the file handle.

Example

```
bReturn = NSW.SetClientFilePtr (hFile, NSW_CURR_FILE_POS, 5)
```

Return Values

True if the method was successful. False if there was an error.

SetCurrentClientFolder

Set a folder as the current folder on the client.

Syntax

```
SetCurrentClientFolder (Foldername)
```

Parameters

Foldername A valid folder name.

Description

The SetCurrentClientFolder method sets the specified folder as the current folder.

Example

```
bReturn = NSW.SetCurrentClientFolder ("C:/mydir");
```

Return Values

True if the method was successful. False if there was an error.

WriteClientFile

Write a string to a file on the client.

Syntax

```
WriteClientFile (Filehandle, Text)
```

Parameters

Filehandle File handle returned by a previous `OpenClientFile()` call.

Text A string to be written to the client file.

Description

The `WriteClientFile` method writes the string specified with the second parameter to the file that is identified by the given file handle. It is written to the current file pointer position.

Example

```
bReturn = NSW.WriteClientFile (hFile, "some nice text");
```

Return Values

True if the method was successful. False if there was an error.